Linear Algebra and its Applications 435 (2011) 1099–1110

Contents lists available at ScienceDirect

Linear Algebra and its Applications

journal homepage: www.elsevier.com/locate/laa

Factoring matrices with a tree-structured sparsity pattern

Alex Druinsky, Sivan Toledo *

School of Computer Science, Tel-Aviv University, Tel-Aviv 69978, Israel

ARTICLE INFO

Article history:
Received 12 May 2010
Accepted 15 March 2011
Available online 13 April 2011

Submitted by V. Mehrmann

AMS classification:
65F50
65F05
15A23

Keywords:
Sibling-dominant pivoting
Tree-structured matrices
Minimum degree ordring
Sparse matrices

ABSTRACT

Let A be a matrix whose sparsity pattern is a tree with maximal degree $d_{\max}$. We show that if the columns of A are ordered using minimum degree on $|A| + |A|^*$, then factoring A using a sparse LU with partial pivoting algorithm generates only $O(d_{\max} n)$ fill, requires only $O(d_{\max} n)$ operations, and is much more stable than LU with partial pivoting on a general matrix. We also propose an even more efficient and just-as-stable algorithm called *sibling-dominant pivoting*. This algorithm is a strict partial pivoting algorithm that modifies the column preordering locally to minimize fill and work. It leads to only $O(n)$ work and fill. More conventional column pre-ordering methods that are based (usually implicitly) on the sparsity pattern of $|A|^* |A|$ are not as efficient as the approaches that we propose in this paper.

1. Introduction[1]

This paper explores the behavior of sparse LU factorizations of matrices whose sparsity pattern is a tree. This class of matrices has received tremendous attention in the literature, primarily because many problems that are very hard or intractable on general matrices can be solved when restricted to this class of matrices. Parter analyzed fill in the Cholesky factorization of tree-structured matrices [19]; his results were greatly extended in the following half-century, but the original sparse Cholesky result focused on tree-structured matrices. Demmel and Gragg show how to efficiently compute the inertia of symmetric tree-structured matrices [8]. Hershkowitz analyzes the D-stability of tree-structured matrices [13]. Klein [16] and Nabben [18] analyze the inverses of tree-structured matrices. Numerous papers analyze the eigenvalues [1,17,21,22], eigenvectors [9], energy [20], and characteristic

* Corresponding author.
E-mail address: stoledo@tau.ac.il (S. Toledo).

[1] A 2-page abstract of this paper, with no proofs and very few experimental results, has been presented in the 2007 *Workshop on Combinatorial Tools for Parallel Sparse Matrix Computations*.

0024-3795/$ - see front matter © 2011 Elsevier Inc. All rights reserved.
doi:10.1016/j.laa.2011.03.035

polynomials [14] of the Laplacians or adjacency matrices of trees. The above citations are just a selection of a large body of results concerning these matrices.

Our paper is similarly motivated. It addresses fundamental questions in sparse Gaussian elimination: how to minimize fill and element growth. Our results, which cover tree-structured matrices, are sharper than corresponding results for more general classes of matrices. In some cases, our theoretical results agree with experimental evidence on general sparse matrices.

The paper begins with an analysis of fill, work, and stability of the factorization when the columns of the matrix are pre-ordered using a minimum-degree algorithm. This analysis constitutes Section 3. The section that follows, Section 4, describes an algorithm that is even more efficient. This algorithm uses strict partial pivoting and a coarse preordering of the columns, but the final column ordering is determined dynamically by inspecting the numerical values in the reduced matrix. This algorithm performs work that is only linear in the order of the matrix and generates only a linear amount of fill. Section 5 shows that our new algorithm is, indeed, highly efficient, more than elimination based on a minimum-degree ordering. The experimental results also show that both algorithms are more efficient than a conservative column preordering algorithm. The notation for the paper and one important backround fact are described in Section 2 and our conclusions from this research in Section 6.

2. Notation and background

Let B be an arbitrary n-by-n square matrix, possibly unsymmetric. The *graph* G_B *of* B is an undirected graph with vertices $\{1, 2, \ldots, n\}$ and with the edge set

$$E_B = \{(i, j) : i \neq j \text{ and } (B_{ij} \neq 0 \text{ or } B_{ji} \neq 0)\}.$$

In an undirected graph G, we denote by $d(i)$ the degree of vertex i, and by $d_{\max}$ the maximal degree in the graph. If $d(i) = 1$, we denote by $p(i)$ the neighbor of i.

Consider an elimination algorithm, say Gaussian elimination. We view the elimination algorithm in two different ways. The algebraic view is that the elimination of k rows and columns produces partial factors and a reduced matrix $B^{(k)}$. The graphical view is that the elimination is a game that defines rules on how to eliminate vertices from a graph. For example, the rule for the Cholesky factorization of a symmetric positive definite matrix is that step k removes vertex k and turns its neighbors into a clique. We denote by $G_B^{(k)}$ the graph that the game produces after the elimination of vertices $1, \ldots, k$ from $G = G_B^{(0)}$. When the identity of B is also clear from the context, we denote $G^{(k)} = G_B^{(k)}$. Under this definition, the graph of $B^{(k)}$ is a subgraph of $G_B^{(k)}$, and it may be a proper subgraph if cancellations occurred.

The vertices of $G_B^{(k)}$ are $\{k + 1, \ldots, n\}$. That is, in the graphical representation we always view vertices $1, \ldots, k$ as having been eliminated first, even if the elimination ordering performed some pivoting steps.

In this paper A always denotes an n-by-n sparse matrix whose graph is a tree.

The results in the paper rely on one piece of backgrond information concerning sparse LU with partial pivoting. The LU factorization algorithm can be implemented in a way that ensures that the total number of operations in the algorithm is proportional to the number of arithmethic operations on nonzeros [12]. This allows us to asymptotically bound the total amount of work in the factorization (work that includes arithmetic but also pointer manipulation and so on) by simply bounding the number of arithmetic operations.

3. *LU* with partial pivoting using minimum degree on *G*

This section analyzes the behavior of the conventional sparse LU with partial pivoting under effective column orderings.

We begin with an analysis of work and fill under the most obvious column ordering, which always eliminates in step k a leaf in $G^{(k-1)}$. We show that $G^{(k)}$ is always a tree, that we can compute this ordering

before the factorization begins, and that this process requires $O(d_{\max}n)$ arithmetic operations and generates $O(d_{\max}n)$ fill. The ordering induced by this rule is simply the minimum degree ordering [10] applied to G. The key observation is that although pivoting can generate fill when G is a tree, the fill can only occur in U; but pivoting essentially cannot change the graph of the reduced matrix.

Lemma 3.1. *Let* 1 *be a leaf of* $G^{(0)}$. *After one step of Gaussian elimination with partial pivoting, the following properties hold:*

(1) $G^{(1)}$ *is a subgraph of* $G^{(0)}$ *induced by vertices* $2, \ldots, n$, *and hence it is also a tree.*
(2) *The first row of* U *contains at most* $d(p(1)) + 1$ *nonzeros.*
(3) *The first column of* L *contains at most two nonzeros.*
(4) *The first elimination step performs at most one comparison, one division, and at most* $d(p(1))$ *multiply–add operations.*

Proof. Since 1 is a leaf, the first column and row of A contain at most two nonzeros, in positions 1 and $p(1)$. If the diagonal element is larger in absolute value, the algorithm simply modifies one diagonal element in $A^{(1)}$ without producing any fill. In this case, the first row of U and the first column of L will each have at most two nonzeros, in positions 1 and $p(1)$. If, on the other hand, $|A_{p(1),1}| > |A_{1,1}|$, Gaussian elimination with partial pivoting will exchange rows 1 and $p(1)$. The algorithm will then subtract a scaled multiple of $A_{p(1),:}$ from $A_{1,:}$, leading to fill of at most $d(p(1)) + 1$ nonzeros in $A_{1,:}$. The columns of these potential nonzeros include columns 1 and $p(1)$, so the new structure of row 1 is exactly the structure of row $p(1)$. This proves the first claim of the lemma.

When pivoting occurs, the structure of the first row of U is the structure of $A_{p(1),:}$. The structure of the first column of L is always the structure of $A_{:,1}$. This proves the other claims in the lemma. □

This lemma leads to the the main result on *LU* with partial pivoting.

Lemma 3.2. *Choose some root for* G *and use it to define the parent* $p(i)$ *of every vertex* i *except the root. Let* Q *be the permutation matrix of any ordering of* $\{1, \ldots, n\}$ *such that* $p(i)$ *is ordered after* i. *If we apply Gaussian elimination with partial pivoting to compute* $Q^TAQ = PLU$, *then*

(1) L *contains at most* $2n$ *nonzeros.*
(2) U *contains at most*

$$\sum_{i=1}^{n}\left(\frac{(d(i)+1)(d(i)+2)}{2} - 3\right)$$

nonzeros.
(3) U *contains at most* $O(d_{\max}n)$ *nonzeros.*
(4) *The algorithm performs* $n - 1$ *comparisons and* $O(d_{\max}n)$ *arithmetic operations.*

Proof. The definition of Q along with Lemma 3.1 ensures that in the kth step, when the algorithm eliminates column k of $(Q^TAQ)^{(k-1)}$, this column has at most two nonzeros (k is a leaf in $G^{(k-1)}$). Consider the rows corresponding to the nonzeros in column k of $(Q^TAQ)^{(k-1)}$. In one of these there are at most two nonzeros (again because k is a leaf in $G^{(k-1)}$). In the other the number of nonzeros is bounded by $d + 1$, where d is the degree of the parent of k in $G^{(k-1)}$.

The bound on the number of nonzeros in L is, therefore, trivial.

The first bound on the number of nonzeros in U is derived by charging the nonzeros in row k of U to $i = p(k)$. The number of nonzeros that i is charged for is the sum over its children of the number of remaining children plus one when they are eliminated. Therefore, i is charged for $d(i) + 1$ nonzeros when its first child is eliminated, for $d(i)$ when the second child is eliminated, and so on, down to 3 nonzeros when the last child is eliminated. This gives the formula inside the summation. Note that a leaf i of G contributes 0 to the sum. The asymptotic bound is easy to prove: no row of U contains more than $d_{\max} + 1$ nonzeros.

The bound on the number of nonzeros in a row of U, along with the fact that columns of L have at most two nonzeros, give the bound on arithmetic operations and comparisons. □

It is possible to construct a family of matrices that shows that the bounds in this lemma are asymptotically tight. We omit the details, but we do describe in Section 5 experimental results with such matrices, which clearly show the tightness of the results.

We note that although this ordering method is fairly natural for tree-structured matrices, it is different than orderings that we would get from ordering algorithms designed for general sparse Gaussian elimination with partial pivoting. One such algorithm is column minimum degree [11] or its approximate-degree variants [6,7]. This algorithm produces minimum-degree-type orderings for $|A|^* |A|$ but without computing the nonzero structure of $|A|^* |A|$. The degree of a vertex in the graph of $|A|^* |A|$ can reach the number of vertices within distance 2 of it in G_A. Therefore, a column minimum degree algorithm may order internal vertices of G_A before leaves in G_A. Our algorithm does not.

Another algorithm for ordering the columns of matrices for fill reduction in *LU* with partial pivoting relies on wide separators [3]. Consider a regular degree-d rooted tree. We can partition hierarchically with wide separators as follows. The root and its children constitute the top-level separator, connected components in the next two levels form the next-level separators, and so on. There are $d + 1$ vertices in each separator: a vertex and its children in G. The wide-separators ordering algorithm does not specify the ordering within separators, so the vertex might be eliminated before its children. This leads to $O(dn)$ nonzeros in L, $O(dn)$ nonzeros in U, and $O(d^2 n)$ arithmetic operations. The number of operations is a factor of d worse than in our approach.

The next lemma shows that when Gaussian elimination with partial pivoting always eliminates a leaf of the reduced graph (as is the case in this algorithm and in the algorithm of the next section), the growth factor in the elimination is limited to $d_{\max} + 1$. Unless $d_{\max}$ is huge, this ensures that the elimination is backward stable.

Lemma 3.3. *Let M be a bound on the magnitude of the elements of A, and let L and U be the triangular factors produced by Gaussian elimination with partial pivoting of AQ for some permutation matrix Q. If the elimination always eliminates leaves in the reduced graph, then the elements of U and of the reduced matrices are bounded in magnitude by $(d_{max} + 1)\, M$ (the elements of L are always bounded in magnitude by 1 in Gaussian elimination with partial pivoting).*

Proof. Off-diagonal elements in the reduced matrices never grow. If we eliminate a leaf i without exchanging the row of the leaf and of its parent, then the only element that is modified in the reduced matrix is the diagonal element associated with the parent. If we do exchange the two rows, then off-diagonals in row $p(i)$ in the new reduced matrix are computed by $A^{(i)}_{p(i),j} = -L_{p(i),i} A^{(i-1)}_{p(i),j}$. Since $|L_{p(i),i}| \leqslant 1$, the off-diagonals cannot grow in magnitude.

Diagonal elements can grow. In an elimination step without a row exchange, we have

$$A^{(i)}_{p(i),p(i)} = A^{(i-1)}_{p(i),p(i)} - L_{p(i),i} A^{(i-1)}_{i,p(i)}.$$

In an elimination step with a row exchange, we have

$$A^{(i)}_{p(i),p(i)} = A^{(i-1)}_{i,p(i)} - L_{p(i),i} A^{(i-1)}_{p(i),p(i)}.$$

Since every diagonal element in the reduced matrices undergoes at most $d_{\max}$ modifications, it is easy to see by induction that elements in the reduced matrices and therefore in U are bounded in magnitude by $(d_{\max} + 1)\, M$. □

If A is not only tree-structured but also tridiagonal and no column pivoting is used, then the proof of this lemma proves a normwise growth bound by a factor of 2. This is a known result that is a special case of the analysis of Bohte [2] of growth in Gaussian elimination with partial pivoting of banded matrices (see also [15, Section 9.5]).

4. Sibling-dominant pivoting

We now show that we can reduce the work and fill to $O(n)$ using a more sophisticated ordering strategy. In this algorithm, the column ordering depends on the numerical values of A, not only on the structure of G and the reduced graphs. Furthermore, we build this ordering dynamically as the algorithm progresses, not as a preprocessing step. But even with the cost of the dynamic ordering taken into account, the algorithm still performs only $O(n)$ operations.

Definition 4.1. The *dominance* of column j in a matrix B is

$$\mathrm{dom}(j) = \begin{cases} \infty & |B_{jj}| \geqslant \max_{i \neq j} |B_{ij}|, \\ \dfrac{\max_{i \neq j} |B_{ij}|}{|B_{jj}|} & \text{otherwise.} \end{cases}$$

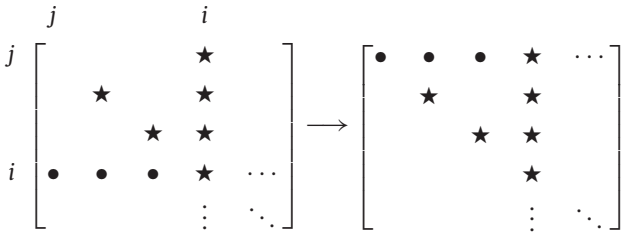
The dominance is not continuous in B_{jj}; as B_{jj} grows, the dominance shrinks towards 1, but then jumps to ∞. We say that column j *dominates* column k if $\mathrm{dom}(j) \geqslant \mathrm{dom}(k)$.

Our algorithm works as follows. It selects a non-leaf vertex i with at most one non-leaf neighbor. It will eliminate next the leaves $\{j_1, \ldots, j_\ell\}$ whose neighbor is i, but in a specific order. To determine the ordering of $\{j_1, \ldots, j_\ell\}$, the algorithm first computes the dominance of $\{j_1, \ldots, j_\ell\}$. Next, the algorithm finds the column in $\{j_1, \ldots, j_\ell\}$ with the largest finite dominance (if any). This column is ordered after all the columns with infinite dominance and before all the other columns with finite dominance. Now the algorithm performs the elimination of $\{j_1, \ldots, j_\ell\}$, breaking ties by not pivoting. That is, if $\left|A^{(k)}_{j,j}\right| = \left|A^{(k)}_{i,j}\right|$ for some $j \in \{j_1, \ldots, j_\ell\}$, the algorithm uses the diagonal element of the reduced matrix as a pivot.

If there is no non-leaf i with at most one non-leaf neighbor, then either the original tree contained just two leaves, or the reduced matrix is now 1-by-1. In both cases the factorization is trivial.

Lemma 4.2. *The algorithm given above performs at most one row exchange (row pivoting step) during the elimination of* $\{j_1, \ldots, j_\ell\}$.

Proof. The elimination sequence starts with columns with infinite dominance (perhaps none). There are two kinds of such columns: columns that are diagonally dominant, and columns with a zero diagonal. If we eliminate a leaf column j with a zero diagonal, rows j and $i = p(j)$ are simply exchanged and then row and column j are dropped from the reduced matrix. In this case, no more pivoting can occur during the elimination of the remaining columns in $\{j_1, \ldots, j_\ell\}$. The diagram below presents this pivoting step; empty spaces represent zeros, • represents a nonzero, and ★ represents a value that is either zero or nonzero.



In the elimination of a leaf column that is diagonally dominant (even weakly dominant), the algorithm does not pivot, and we only modify element $A_{i,i}$ in the reduced matrix. Because only this element is modified, the dominance of the remaining columns in $\{j_1, \ldots, j_\ell\}$ is preserved.

Now consider what happens when we eliminate the first column j in $\{j_1, \ldots, j_\ell\}$ with a finite dominance. Its dominance is larger than that of any other column in $\{j_1, \ldots, j_\ell\}$ with a finite dominance.

Because j has finite dominance, the algorithm pivots in column j; it exchanges rows j and $i = p(j)$. In the reduced matrix, we have

$$A_{i,k}^{(j)} = A_{j,k}^{(j-1)} - \frac{A_{j,j}^{(j-1)}}{A_{i,j}^{(j-1)}} A_{i,k}^{(j-1)}.$$

(This formula represents both the row exchange and the numerical modification.) For a sibling k of j, we have $A_{j,k}^{(j-1)} = 0$, so

$$A_{i,k}^{(j)} = -\frac{A_{j,j}^{(j-1)}}{A_{i,j}^{(j-1)}} A_{i,k}^{(j-1)}$$

and the absolute values satisfy

$$\begin{aligned} \left|A_{i,k}^{(j)}\right| &= \left|\frac{A_{j,j}^{(j-1)}}{A_{i,j}^{(j-1)}}\right| \left|A_{i,k}^{(j-1)}\right| \\ &= \frac{\left|A_{i,k}^{(j-1)}\right|}{\mathrm{dom}(j)}. \end{aligned}$$

We claim that the remaining uneliminated columns in $\{j_1, \ldots, j_\ell\}$ have become diagonally dominant. Let k be one of these columns. We have $\mathrm{dom}(j) \geqslant \mathrm{dom}(k)$, so

$$\begin{aligned} \left|A_{k,k}^{(j)}\right| &= \left|A_{k,k}^{(j-1)}\right| \\ &= \frac{\left|A_{k,k}^{(j-1)}\right|}{\left|A_{i,k}^{(j-1)}\right|} \left|A_{i,k}^{(j-1)}\right| \\ &= \frac{\left|A_{i,k}^{(j-1)}\right|}{\mathrm{dom}(k)} \\ &\geqslant \frac{\left|A_{i,k}^{(j-1)}\right|}{\mathrm{dom}(j)} \\ &= \left|A_{i,k}^{(j)}\right|. \end{aligned}$$

This implies that if we eliminate k next, the algorithm will not pivot. Since we have already shown that when the algorithm does not pivot, it does not modify other sibling columns, the other siblings will remain diagonally dominant and they will not require pivoting either. □

We note that columns with infinite dominance can be ordered after the column with the largest finite dominance, but the proof becomes longer. Since there is no benefit in this variant ordering, we kept the ordering strict and the proof short.

In our sibling-dominant elimination, the elimination of each column uses the regular partial-pivoting rule. It appears that the column ordering cannot be computed before the numerical factorization begins. Nonetheless, the overall effort to compute the column ordering is $O(n)$ operations.

Lemma 4.3. *The total amount of work to order the columns in sibling-dominant partial pivoting requires only $O(n)$ operations and $O(n)$ storage.*

Proof. We pre-compute a partial column ordering before the numerical factorization begins and refine it into a total ordering during the numerical factorization. We use two integer vectors of size n: `column-index` and `partial-order`.

The preordering phase begins with an arbitrary choice of root for G_A, say vertex 1. We write the index of the root into `column-index[1]` and an invalid column index (such as -1 or $n+1$) into `partial-order[1]`. We now perform a breadth-first search (BFS) of G_A starting from the root. When we visit a new vertex, we write its column index into the next-unused location in `column-index` and the index of its parent in the rooted G_A into the first unused location in `partial-order`. The identity of the parent is always known, because it is the vertex from which we discovered the current one.

The partial ordering is specified by the reverse ordering of the same-parent groups of columns in the two vectors. That is, the first group of columns to be eliminated will be the last group with the same parent to be discovered by the BFS. It is not hard to see that if the elimination ordering respects this partial order, then each group with the same parent is eliminated when all the members of the group are leaves of the reduced graph.

Given the vectors `column-index` and `partial-order` we can enumerate the siblings' groups in time proportional to their size, by scanning from the last uneliminated column in the vectors towards the beginning of the vectors until we find a column with a different parent (or no parent). During this enumeration, we can find and eliminate all the columns with infinite dominance, and we can compute the dominance of the rest. Once this traversal and the elimination of infinite-dominance columns is complete, we eliminate the column with the largest finite dominance, if any, and then we eliminate its siblings.

Clearly, the total work and storage required for computing the column ordering is $O(n)$. The keys to efficiently compute the ordering are an efficient pre-computation of the sibling groups (using BFS) and the fact that in each group we only need to find one dominant sibling, not to sort all the siblings by dominance. □

The sibling-dominant column ordering results in a partial-pivoting Gaussian elimination algorithm that only performs $O(n)$ work and only generates $O(n)$ fill.

Theorem 4.4. *The work and fill in sibling-dominant partial pivoting are both $O(n)$.*

Proof. Sibling-dominant partial pivoting is a special case of partial-pivoting using a minimum degree ordering. By Lemmas 3.1 and 3.2, each column in L contains at most two nonzeros and requires a constant number of operations to compute. A column j with $\mathrm{dom}(j) = \infty$ can either have a zero diagonal element or a diagonal element that is dominant in the classical sense ($|A_{jj}^{(k)}| \geqslant \max_{i \neq j} |A_{ij}^{(k)}|$). When we eliminate an inifinite-dominance column with a nonzero diagonal, no pivoting is required. In this case, the row of U is simply the row of the reduced matrix, which contains at most two nonzeros because the reduced graph is a tree and the eliminated column is a leaf. When we eliminate a column with infinite dominance and a zero diagonal element, or when there are no such columns and we eliminate the column with the largest finite dominance, we exchange two rows. This causes fill in the row of U: the number of fill elements and the number of multiply–add operations is bounded by the number of remaining siblings in the group plus 2. However, once we perform such a row exchange, further eliminations in the same sibling group will not require any row exchanges (Lemma 4.2). This implies that no more fill will occur in U within this group, and that the number of arithmetic operations per remaining column in the group is bounded by a small constant. Since we showed that the ordering of the elimination steps also requires only $O(n)$ operations, the lemma holds. □

5. Experimental results

We conducted three sets of experiments in order to demonstrate the effectiveness of the sibling-dominant pivoting method. In all the experiments we used almost-complete regular trees. These trees are degree-$d_{\max}$ complete trees with some of the leaves removed to obtain a specific number of vertices. In these trees, the degree of all the internal vertices is the same except for one or two (the root and perhaps one vertex in the second-to-last level). Fig. 5.1 shows an example of such a tree.

The first experiment, whose results are shown in Figs. 5.2–5.4, used trees with 1000 vertices and $d_{\max}$ that ranged from 2 to 999. The matrices are all symmetric and the value of all the nonzero

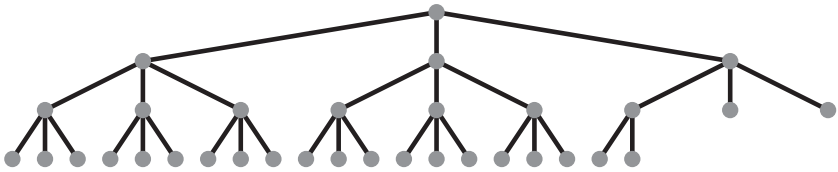


Fig. 5.1. An almost complete regular tree with $d_{\max} = 4$.

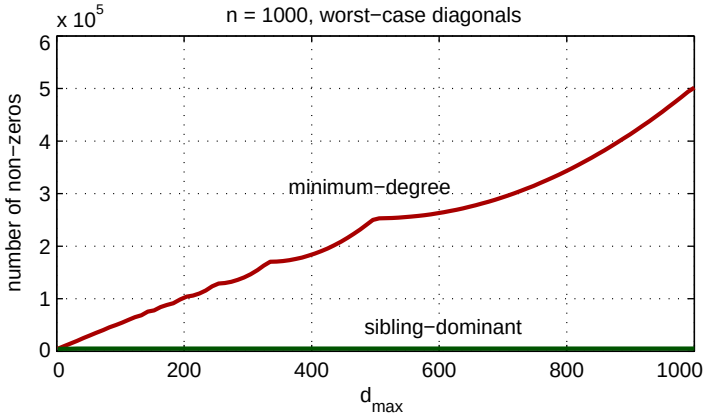


Fig. 5.2. The number of nonzeros in the *LU* factors of symmetric matrices whose graphs are almost-complete regular trees. The values of the elements of the matrices were chosen so as to cause as much fill as possible in *LU* with partial pivoting when the elimination ordering is bottom-up.

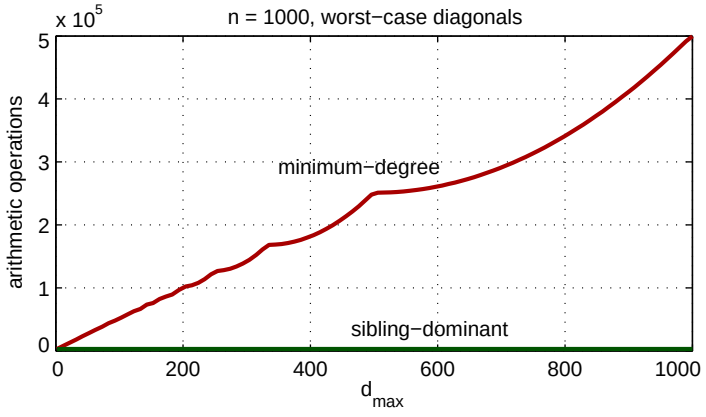


Fig. 5.3. The number of arithmetic operations in the factorizations, on the same matrices as in Fig. 5.2.

off-diagonal elements is 1. The values of the diagonal elements are constructed so as to cause *LU* with partial pivoting to fill as much as possible when the column ordering is produced by a minimum-degree algorithm applied to $|A| + |A|^*$. In other words, to construct the matrices we first construct their graph, we then construct a minimum-degree ordering for this graph, permute the matrix according to this ordering, and finally compute the values of the diagonal elements.

The results of the experiments show clearly that the amount of fill and the number of arithmetic operations that *LU* with partial pivoting performs on these matrices is linear in $d_{\max}$, even though the elimination was ordered bottom-up using a minimum-degree algorithm. The slight non-linearities are

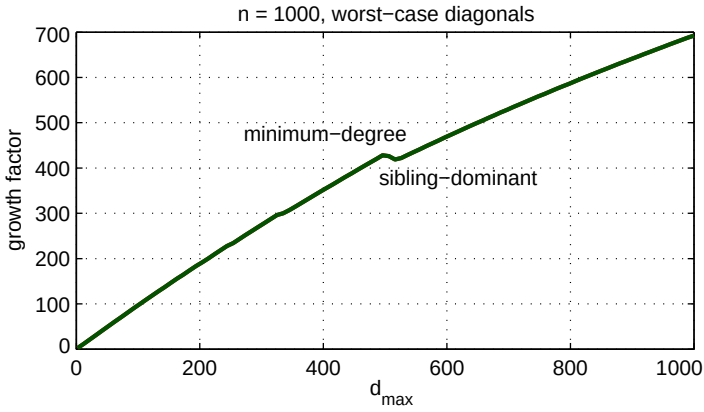


Fig. 5.4. The growth in U on the same matrices as in Fig. 5.2. The two data sets overlap each other exactly.

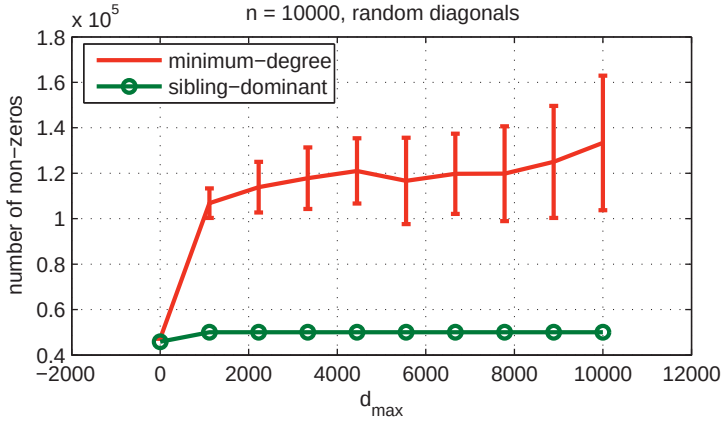


Fig. 5.5. The number of nonzeros in the LU factors of symmetric matrices whose graphs are almost-complete regular trees. The values of the off-diagonal nonzeros are all 1 and the values of the diagonal elements are random. The markers and the lines show the average of 100 matrices and the error bars show the standard deviation.

caused by the fact that the trees are not complete. The growth in both algorithms is exactly the same and appears to be slightly sub-linear. The fact that the growth in the two algorithms is exactly the same is an artifact of the special structure of these matrices; in general, the growth factors would be different.

In the second set of experiments we used random diagonal elements with uniform distribution in $\left[\frac{1}{2}, 1\right]$; the non-zero off-diagonals were still all 1. The trees were still almost complete with 10,000 vertices. For each of 10 different values of $d_{\max}$ we generated 100 random matrices. The results, shown in Figs. 5.5–5.7 indicate that on these trees, partial pivoting with a bottom-up column preordering fills much less than the worst-case bound. The average amount of fill grows very little with $d_{\max}$, but the variance in fill does grow with $d_{\max}$. Still, in absolute terms the amount of fill and arithmetic in sibling-dominant pivoting is much smaller than in partial pivoting. The growth in U is exactly linear in $d_{\max}$ and is almost completely invariant to the random choice of diagonal elements. In this case too, the growth in the two algorithms is the same (again an artifact of the special construction).

The last set of experiments used similar matrices, but the value of all the diagonal elements was n (number of vertices and the order of the matrices). These matrices are strongly diagonally dominant. The LU factorization with partial pivoting was computed after a column-preordering phase that started

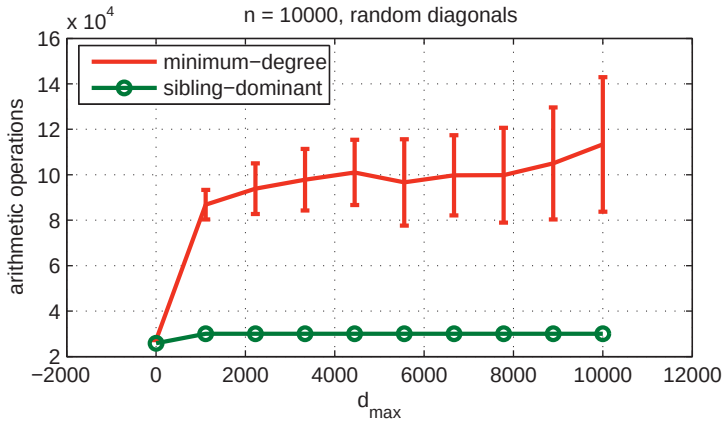


Fig. 5.6. The number of arithmetic operations in the experiments described in Fig. 5.5.

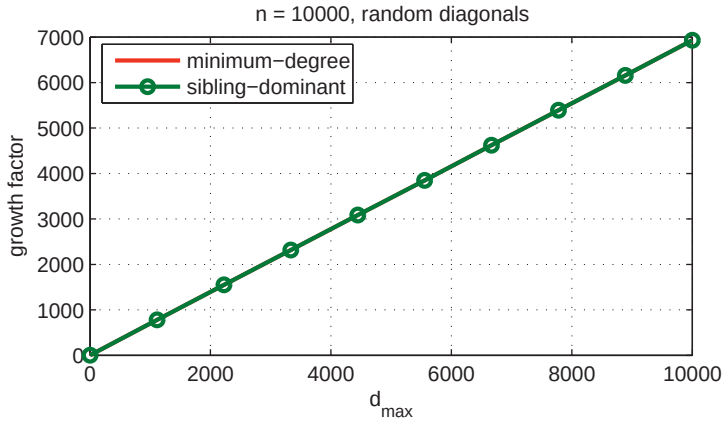


Fig. 5.7. The growth factors in the experiments described in Fig. 5.5.

by a random column ordering but then reordered again using COLAMD [6], a minimum-degree heuristic ordering that tries to minimize fill in the Cholesky factor of $|A|^* |A|$ (without explicitly computing $|A|^* |A|$ or its nonzero structure). Multiple initial random orderings of each matrix produced variance in the fill and work of the partial-pivoting code, even with the COLAMD reordering. The matrices are diagonally dominant, so no pivoting was performed by *LU* with partial pivoting. Therefore, minimum degree on $|A| + |A|^*$ would have produced no fill and linear work.

The results in Figs. 5.8 and 5.9 indicate that COLAMD leads to superlinear fill and work in *LU* with partial pivoting even though the matrix is a diagonally dominant tree. COLAMD indeed guards the factorization from catastrophic fill, by considering any pivoting sequence. But the price of this conservatism is some fill even when no pivoting is performed.

6. Conclusions

This paper explored the behavior of *LU* with partial pivoting on matrices whose sparsity pattern is a tree.

We have argued and demonstrated experimentally that a conservative column pre-ordering that attempts to minimize fill for any pivoting sequence, such as COLAMD, can lead to fill and work that

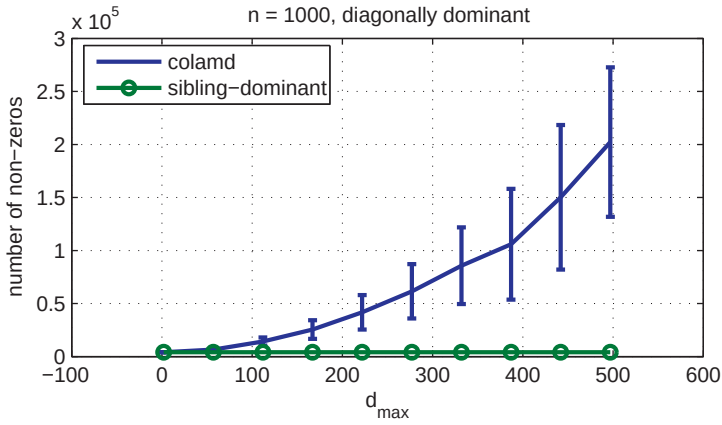


Fig. 5.8. The number of nonzeros in the factorization of symmetric diagonally-dominant tree-structured matrices. For *LU* with partial pivoting, the matrices were ordered using COLAMD following an initial random ordering; the randomness in the results is caused by this initial random ordering.

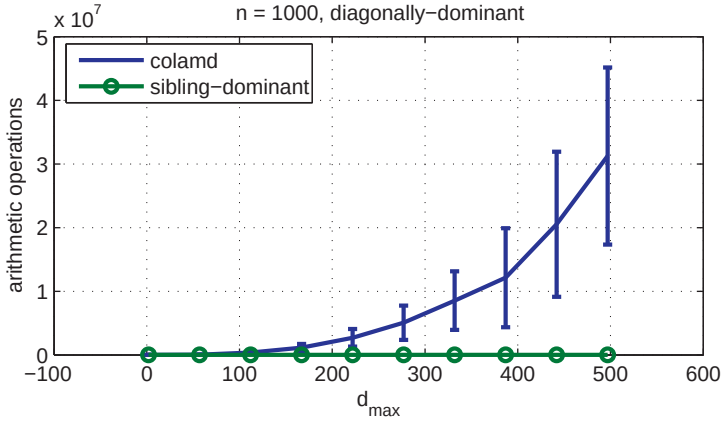


Fig. 5.9. The number of arithmetic operations in the experiments shown in Fig. 5.8.

are superlinear in the maximal degree in the tree. We have shown theoretically that a column preordering that is obtained from a minimum-degree algorithm that is applied directly to $|A| + |A|^*$ guarantees fill and work that are only linear in the maximal degree. We have argued and shown experimentally that this bound is tight. Finally, we have proposed a more dynamic pivoting rule called sibling-dominant pivoting. This pivoting rule uses strict partial pivoting for stability, but performs local column reorderings to minimize fill and work. The amount of work and fill in our new algorithm is linear in the order of the matrix, independently of the structure and maximal degree of the tree.

We have also analyzed the growth in factorizations based on both minimum-degree on $|A| + |A|^*$ and sibling-dominant pivoting. We have shown that it is bounded in both cases by $d_{\max} + 1$, a much smaller bound than the 2^{n-1} bound for general *LU* with partial pivoting.

These results have two consequences whose significance may transcend the class of tree-structured matrices. First, the results show that on some restricted classes of matrices, orderings based on the structure of $|A| + |A|^*$ may be provably better than more conservative orderings based on the structure of $|A|^* |A|$, even in factorizations with pivoting. Second, they show that dynamic but cheap-to-compute local column reorderings can dramatically reduce fill and work. This was known experimentally from

the experiences gathered by the UMFPACK 4 algorithm [4,5], but our results are the first theoretical ones in this area.

We did not find a symmetric elimination method for tree-structured matrices that is stable and as efficient as sibling-dominant pivoting (which does not require symmetry). We did find a method with $O(d_{\max} n)$ fill and $O(d_{\max}^2 n)$ work, but it is clearly not competitive with sibling-dominant pivoting, so there is little reason to use it for solving linear systems of equations. We omit the details. Symmetric factorizations are also useful for computing the inertia of a matrix (which cannot be computed from an *LU* factorization), but the inertia of tree-structured matrices can be computed using a sparse LDL^* factorization with no pivoting [8].

Acknowledgements

We thank the two anonymous referees for suggestions and comments that helped us improve the paper. This research was supported by an IBM Faculty Partnership Award, by grants 848/04 and 1045/09 from the Israel Science Foundation (founded by the Israel Academy of Sciences and Humanities), and by grant 2002261 from the United States–Israel Binational Science Foundation.

References

[1] Türker Bıyıkoğlu, Josef Leydold, Semiregular trees with minimal Laplacian spectral radius, Linear Algebra Appl. 432 (9) (2010) 2335–2341. Special Issue devoted to Selected Papers presented at the Workshop on Spectral Graph Theory with Applications on Computer Science, Combinatorial Optimization and Chemistry (Rio de Janeiro, 2008).
[2] Zvonimir Bohte, Bounds for rounding errors in the Gaussian elimination for band systems, J. Inst. Math. Appl. 16 (2) (1975) 133–142.
[3] Igor Brainman, Sivan Toledo, Nested-dissection orderings for sparse LU with partial pivoting, SIAM J. Matrix Anal. Appl. 23 (2002) 998–1012.
[4] Timothy A. Davis, Algorithm 832: UMFPACK v4.3—an unsymmetric-pattern multifrontal method, ACM Trans. Math. Software 30 (2) (2004) 196–199.
[5] Timothy A. Davis, A column pre-ordering strategy for the unsymmetric-pattern multifrontal method, ACM Trans. Math. Software 30 (2) (2004) 165–195.
[6] Timothy A. Davis, John R. Gilbert, Stefan I. Larimore, Esmond G. Ng, Algorithm 836: COLAMD, a column approximate minimum degree ordering algorithm, ACM Trans. Math. Software 30 (3) (2004) 377–380.
[7] Timothy A. Davis, John R. Gilbert, Stefan I. Larimore, Esmond G. Ng, A column approximate minimum degree ordering algorithm, ACM Trans. Math. Software 30 (3) (2004) 353–376.
[8] James W. Demmel, William Gragg, On computing accurate singular values and eigenvalues of matrices with acyclic graphs, Linear Algebra Appl. 185 (1993) 203–217.
[9] Miroslav Fiedler, Eigenvectors of acyclic matrices, Czechoslovak Math. J. 25 (100) (1975) 607–618.
[10] A. George, J.W.H. Liu, The evolution of the minimum-degree ordering algorithm, SIAM Rev. 31 (1989) 1–19.
[11] John R. Gilbert, Cleve Moler, Robert Schreiber, Sparse matrices in MATLAB: design and implementation, SIAM J. Matrix Anal. Appl. 13 (1) (1992) 333–356.
[12] John R. Gilbert, Tim Peierls, Sparse partial pivoting in time proportional to arithmetic operations, SIAM J. Sci. Stat. Comput. 9 (1988) 862–874.
[13] Daniel Hershkowitz, Stability of acyclic matrices, Linear Algebra Appl. 73 (1986) 157–169.
[14] Abbas Heydari, Bijan Taeri, On the characteristic and Laplacian polynomials of trees, Linear Algebra Appl. 432 (2–3) (2010) 661–669.
[15] Nicholas J. Higham, Accuracy and Stability of Numerical Algorithms, second ed., Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2002.
[16] D.J. Klein, Treediagonal matrices and their inverses, Linear Algebra Appl. 42 (1982) 109–117.
[17] Jianxi Li, Wai Chee Shiu, An Chang, On the *k*th Laplacian eigenvalues of trees with perfect matchings, Linear Algebra Appl. 432 (4) (2010) 1036–1041.
[18] Reinhard Nabben, On Green's matrices of trees, SIAM J. Matrix Anal. Appl. 22 (4) (2001) 1014–1026.
[19] S. Parter, The use of linear graphs in Gauss elimination, SIAM Rev. 3 (1961) 119–130.
[20] María Robbiano, Raúl Jiménez, Improved bounds for the Laplacian energy of Bethe trees, Linear Algebra Appl. 432 (9) (2010) 2222–2229. Special Issue devoted to Selected Papers presented at the Workshop on Spectral Graph Theory with Applications on Computer Science, Combinatorial Optimization and Chemistry (Rio de Janeiro, 2008).
[21] P. Rowlinson, On multiple eigenvalues of trees, Linear Algebra Appl. 432 (11) (2010) 3007–3011.
[22] Shang-Wang Tan, On the weighted trees with given degree sequence and positive weight set, Linear Algebra Appl. 433 (2) (2010) 380–389.